Representation Theorems and Primitive Predicates for Logic Programs

AUTHOR(S):

YOKOMORI, Takashi

CITATION:

YOKOMORI, Takashi. Representation Theorems and Primitive Predicates for Logic Programs. 数理解析研究所講究録 1986, 586: 1-17

ISSUE DATE:

1986-03

URL:

http://hdl.handle.net/2433/99394

数理解析研究所講究録
586巻 1986年 1-17

Representation Theorems and Primitive Predicates for Logic Programs

by

Takashi YOKOMORI
横森 貴

International Institute for Advanced Study of Social Information Science(IIAS-SIS), Fujitsu Limited
140 Miyamoto, Numazu, Shizuoka 410-03 JAPAN

ABSTRACT

This paper concerns the representation theorem for logic programs in terms of formal grammatical formulation. First, for a given logic program P the notion of the success language of P is introduced. Then, based on this language theoretic characterization of a logic program two representation theorems for logic programs are provided. We show that there effectively exists a fixed logic program with the property that for any logic program one can find an equivalent logic program such that it can be expressed as a conjunctive formula of a simple program and the fixed program. Further, by introducing the concept of an extended reverse predicate, it is shown that for any logic program there effectively exists an equivalent logic program which can be expressed as a conjunctive formula consisting of only extended reverse programs and append programs.

1. Introduction

Since, needless to say the original work of Colmerauer and Kowalski([1] and [7]), a recent world-wide trend on FGCS concep-

tion has been one of the primary subjects, there are numerous work on logic programming languages and the theory of logic programs. It is well accepted that, among others, the research on a subset of first-order predicate logic called Horn clause logic has taken the central position in this area because of its importance of providing an interesting formal computation model for a programming language PROLOG. As is well-known, PROLOG, based on the procedural interpretation to Horn clause logic, has an operational semantics determined by the resolution principle. In the context of the semantics of predicate logic as a programming language van Emden and Kowalski ([3]) have studied on model-theoretic, operational and fixedpoint semantics of logic programs, while using a Turing machine formulation Shapiro ([9]) has defined and argued a kind of model-theoretic semantics of logic programs.

In this paper we are concerned with establishing two representation theorems for "logic programs(Horn clause programs)" in terms of formal language theoretic formulation. In course of the formal grammatical treatment of logic programs we introduce the notion of the success language of a logic program over a finite alphabet, which turns out to be another way of providing a model-theoretic semantics for logic programs. Here, by formal grammars we mean generative grammars of Chomsky, and it should be remarked that the theory of formal languages([e.g.,[5], [6],[8]) has been well-developed enough in itself to make a lot of contributions to other research areas such as the theory of logic programming. This view may be supported, for example, when we think of the similarity between the refutation process in logic programs and the derivation steps in context-free grammars, and note that logic programs can be regarded as a kind of an extention of context-free grammars. In fact, Shapiro investigates the computational complexity of logic programs using the similarity of their operational behaviors to those of alternating Turing machines.([9])

With the help of an encoding technique it is shown that one

can associate a logic program with a formal language (the success language mentioned above) over a finite alphabet. This leads to a semantic characterization of logic programs as previously mentioned, although that is not our primary concern in the current paper. This kind of semantic approach to logic programs has been already preceeded by the paper [11]. It has been shown that any recursively enumerable language can be specified as a conjunctive formula of two deterministic logic programs and one simple logic program that serves as a mapping on the set of words. The work in this paper is motivated by the result above and extends it to examine what operations are primitive in representing logic programs.

In this paper we present two representation theorems for logic progams. We show that there effectively exists a fixed logic program with the property that for any logic program one can find an equivalent logic program such that it can be expressed as a conjunctive formula of a simple program and the fixed program.

Further, by analysing components in the representation result and introducing the concept of "extended reverse", it is also shown that for any logic program one can find an equivalent logic program expressed as a conjunctive formula consisting of only "extended reverse" programs and "append" programs.

This paper is organized as follows. Section 2 is concerned with terminology, basic notions and results needed through the paper. In Section 3 a representation theorem for logic programs is established. Section 4 deals with the problem of what operations (predicates) are primitive for the representation formula obtained in Section 3. Concluding remarks and the future reseach direction are briefly given in Section 5.

2. Preliminaries

2.1 Formal Grammars and Their Languages

We now introduce a generative device which plays the main role in all of subsequent sections in this paper.

Definition.

A generative grammar is an ordered quadruple G = (N, T, P, S_0) where N and T are disjoint finite alphabets, S_0 is in N, and P is a finite set of production rules of the form Q_1 -> Q_2 such that Q_2 is a word over the alphabet V = N$\cup$T and Q_1 is a word over V containing at least one symbol of N. The elements of N are called nonterminals and those of T terminals; S_0 is called the initial symbol.

A word u generates directly a word v, in symbols, u=>v, if and only if there are words u', u", Q_1, Q_2 such that u=u'Q_1u", v=u'Q_2u", and Q_1 -> Q_2 belongs to P. Thus, => is a binary relation on the set V^*(the set of all words over V including empty word e). We denotes V^*-{e} by V^+. Let =>* be the reflexive, transitive closure of =>. The language L(G) generated by G is defined by

L(G) = { w in T^* | S_0 =>* w }.

L(G) is called a language over T (or on T^*).

Grammars are, in general, classified by the form of production rules, which yields a hierarchy of corresponding language families.

Definition.

(1) A generative grammar is also called phrase structure grammar. Let G = (N,T,P,S_0) be a phrase structure grammar. Then, G is called

(i) context-free if each production rule is of the form X -> Q, where X in N, and Q in V^*,

(ii) regular if each production rule is of one of the two form X->a or X->aY, where a in T and X,Y in N, with the possible exception on the production rule S_0 -> e whose occurrence in P implies that S_0 does not occur on the right hand side of any rule in P.

(2) Let G = (N,T,P,S_0) be a context-free grammar with the property that (i) every rule in P is of the form A -> ax, where A in N, a in T, x in N^*,and (ii)for all A in N, a in T, A->ax and

A->ay in P implies x=y. Then, G is called simple deterministic.

Definition.

Let L be a subset of T* for some alphabet T, and let X be in {phrase structure, context-free, simple determinstc, regular}. Then, L is called an X language if L=L(G) for some X grammar G. Further, a language generated by a phrase structure grammar is also called recursively enumerable.

Definition.

Let T be an alphabet. For each a in T, let f(a) be a word (possibly over a different alphabet from T). Then, let f(e) = e, f(xy) = f(x)f(y) (x,y in T^*). The mapping f is extended to the power set of T^* as follows: for each L over T, f(L) = {f(w)| w in L}. The mapping f is called a homomorphism on T^*.

A homomorphism f on T^* is called a weak identity if for each a in T, f(a) is either the symbol a itself or the empty word e.

Definition.

A deterministic generalized sequential machine(dgsm) with accepting states is a 6-tuple A = (Q,T,D,d,q_0,F), where

Q : a finite set of states, T : a finite set of input symbols, D : a finite set of output symbols, d : transition function from $Q \times T$ to $Q \times D^*$, q_0 : the initial state in Q, and F : a subset of Q (a set of final states).

The function d is extended to $Q \times T^*$ as follows: for q in Q, x in T^*, a in T,

d(q,e) = (q,e),

d(q,ax) = (r,y)

where

$y = w_1 w_2$

$d(q,a)=(p,w_1)$, $d(p,x) = (r,w_2)$ for some p in Q, w_1, w_2 in D^*.

Let f_A be a mapping defined by

$f_A(x) = y$ iff $d(q_0,x) = (p,y)$ for some p in F.
The mapping f_A so defined is called a dgsm mapping of A.

Notation.

Let T be a finite alphabet. For a word $w = a_1 \cdots a_n$ $(n \geq 0)$ in T^*, the $(\tilde{})$-version $\tilde{w}$ denotes $\tilde{a}_1 \cdots \tilde{a}_n$. Further, w^R denotes the reverse $a_n \cdots a_1$.

2.2 Logic Programs and Their Languages

This subsection introduces the concepts of a logic program and its associated language we shall deal with in the subsequent sections. We assume the reader to be familiar with the rudiments of mathematical logic.

Definition

A logic program is a finite set of Horn clauses, which are universally quantified logical sentences of the form

$$A \leftarrow B_1,\ldots,B_n \quad (n \geq 0) \qquad (C)$$

where the A and the B's are atomic formulae. In the above clause (C) A is called the clause's head, while B's are called the clause's body. If n = 0, then we simply donote it by A instead of A <-.

Atomic formulae occurring in a logic program are called goals. A program is said to be dominated by a goal if the predicate name of the goal occurs only once as the head of a clause in the program.

[Notational Convention]

(i) We use upper-case letters such as X, Y, Z for variable symbols and lower-case letters such as x, y, z for ground terms. For terms, letters t, s, r are often used. The boldface versions like **P**, **Q** are used for logic programs, while normal upper-case letters like P, Q are used for goals, and lower-case letters p,q for goal names.

(ii) For a logic program dominated by a goal, we sometimes refer to the program in terms of the name of the goal. In such a case it is assumed that the program name is the capital letter P of the goal name "p".

Definition.

Let P be a logic program and Q a goal. If there is a refutation of a goal Q from P, then we say P succeeds on Q, or Q succeeds (in P).

In this paper we are concerned with logic programs whose data domains are finitely generated by a fixed set of symbols.

Definition.

Let P be a logic program. The Herbrand universe of P is the set of all ground terms constructable from the set of constants C and the set of function symbols F occuring in P, and we denote it by D(F,C). Then, a logic program P is called a logic program over C if F comprises only one function symbol, and its Herbrand universe is denoted by D(C).

As shown below, any Herbrand universe for a logic program can be coded in an appropriate manner into the domain D(T) constructed from some fixed finite set of symbols T. In other words, any ground term which possibly appears in a program can be taken as a word over some finite alphabet T.

Lemma 2.1

There exist a fixed finite set of symbols T and a one-to-one mapping f such that for any logic program P with the domain D(F,C) and for any goal $p(X_1,\ldots,X_n)$ there exist a logic program P' with the domain D(T) and a goal p'(X) with the property that P succeeds on $p(x_1,\ldots,x_n)$ iff P' succeeds on p'(x), where $x=f(x_1,\ldots,x_n)$.

Proof.

Let $g_1,g_2,\ldots$ be an enumeration of all function symols

occurring in D(F,C) of P. (Note that a constant k can be taken as a 0-ary function symbol as in k().)

Introduce a mapping c from the set D(F,C) to the set of lists as follows :

for a term $t = g_i(s_1,...,s_m)(m \geq 0)$,

$c(t) = [\%,@^i,\$,c(s_1),...,c(s_m),\$]$.

where "[" and "]" are the list notation ,$,$,@,% are new symbols, and $@^i$ denotes a sequence @,...,@ of i @s.

Further, for an n-tuple of terms $(t_1,...,t_n)$, let f be defined by

$f(t_1,...,t_n)$ = flatten($[c(t_1),\#,...,\#,c(t_n)]$),

where "flatten" is a mapping of flattening lists,

is a new symbol(argument separator).

Define p'(X) as follows :

$p'(X)$ <- $flat(X_1,...,X_n,X)$, $p(X_1,...,X_n)$ --- (C_0)

where $flat(X_1,...,X_n,X)$ succeeds iff $X = f(X_1,...,X_n)$.

Further, let **P'** be **P** $\{C_0\}$. Then, it is easily seen that **P'** succeeds on $p'(f(x_1,...,x_n))$ iff **P** succeeds on $p(x_1,...,x_n)$ for x_i in D(F,C). Let T = {#,$,$,@,%,NIL}, where NIL denotes empty list, then D(T) is the set of lists constructed from T and the unique function symbol of the list constructor. Obviously, this satisfies the desired conditions. □

Thus, it is sufficient for general discussion to deal with only logic programs over some fixed finite set.

[Conventions]

(1) In what follows, it may be assumed that (i)a logic program over T has the domain of all lists constructed from a finite set of constants T, and (ii) otherwise specified, a goal is assumed to be a 1-ary predicate.

(2) As a notation, given finite set of symbols T and a word $w = a_1 \cdots a_n$ on T^*, the boldface **w** denotes the list version $[a_1,...,a_n]$.

Logic programs together with goals are classified by the

types of their associated languages.

Definition.

Let **P** be a logic program over a finite set of symbols T and Q (=q(X)) be a goal in **P**.

(i) A language over T defined by

$$L(\mathbf{P},Q,T) = \{ w \text{ in } T^* \mid \mathbf{P} \text{ succeeds on } q(\mathbf{w}) \}$$

is called the success language of Q in **P**. In this case L(**P**,Q,T) is often denoted by L(**P**,q,T). If **P** is dominated by p(X) or a program "**P**" is named after the goal name "p", then we simply write L(**P**,T) and call it the success language of **P**.

(ii) A logic program **P** is called X if L(**P**,Q,T) is an X language for all goal Q in **P**.

(iii) Let p(X,Y) be a goal dominating **P**, and for x in T^*, let $f_{\mathbf{P}}(x) = \{y \text{ in } T^* \mid \mathbf{P} \text{ succeeds on } p(\mathbf{x},\mathbf{y}) \}$. Then, a logic progam **P** is called

(1) homomorphism if $f_{\mathbf{P}}$ is a homomorphism,

(2) weak identity if $f_{\mathbf{P}}$ is a weak identity,

on T^*.

Finally,

(iv) Let **P** and **P'** be two logic programs over T, and let p(X) and p'(X) be goals in **P**, **P'**, respectively. Then **P** with p(X) and **P'** with p'(X) are equivalent if L(P,p,T) = L(P',p',T).

We end this section with presenting a result showing the expressive capability of logic programs we are dealing with in this paper.

It has been shown in literature ([10],[11]) that for any recursively enumerable language L over T, there exist a logic program **P** over T and a goal Q such that L is the success language of Q in **P**. Conversely, it is shown that for any logic program **P** over T and a goal Q, the success language L(**P**,Q,T) is a recursively enumerable language, which is proved by constructing a Turing machine simulating the resolution process for Q from **P** and accepting the

success language of Q in **P** ([9]). (Note that a language is recursively enumerable if and only if it is accepted by a Turing machine.)
Hence, we have the following :

Theorem 2.1

The class of success languages of logic programs is equal to the class of recursively enumerable languages.

It may be possible to state that the success language of a logic program provides us a kind of model-theoretic semantics (or denotational semantics) for logic programs.

3. A Representation Theorem

In this section a representation theorem for logic programs is presented.

We shall show that there exists a fixed simple program which plays a role of generator for the class of logic programs. Such a program can be obtained by making a slight modification to "reverse" program.

Lemma 3.1

For any recursively enumerable language L over an alphabet T there exist a simple deterministic language Sp on $K^+\tilde{K}^+$ (for some alphabet K including T), and a weak identity h such that $L = h(\{w\tilde{w}^R \mid w \text{ in } K^+\} \cap Sp)$, where $Sp = \{x¢\tilde{f}\tilde{y}^R \mid f(x)=y\}$, f is a dgsm mapping of $A = (Q,K,D,d,q_0,F)$ depending on L, h preserves the alphabet of L and erases other symbols, ¢ is a new symbol.
(See Theorem 11 in [4])

Theorem 3.1(Representation Theorem 1)

Let T be a fixed alphabet. Then, there exists a fixed logic program $\mathbf{R_0}$ with the property that for any logic program **P** over T with a goal p(X) one can find an equivalent logic program **P'** with

a goal p'(X) such that it can be expressed by

$$p'(X) \leftarrow r_0(X,Y),\ s_P(Y) \qquad (3\text{-}1)$$

for some simple deterministic program $\mathbf{S_P}$.

Proof.

From Theorem 2.1 and Lemma 3.1, for any logic program **P** over T with a goal p(X) there is a simple deterministic language Sp on $K^+\tilde{K}^+$ and a weak identity h such that $L(\mathbf{P},p,T) = h(\{w\tilde{w}^R \mid w \text{ in } K^+\} \cap Sp)$, where $Sp = \{x\not\tilde{\not}\tilde{y}^R \mid f(x)=y\}$, f is a dgsm mapping of $A = (Q,K,D,d,q_0,F)$ depending on $L(\mathbf{P},p,T)$, and h(a) = a (for all a in T), h(a) = e (otherwise).

Construct three logic programs so that $\mathbf{M_T}$, $\mathbf{I_T}$, and $\mathbf{S_P}$ may determine the language $M_0(=\{w\tilde{w}^R \mid w \text{ in } K^+\})$, h, and Sp, respectively.

(1) $\mathbf{M_T}$ is defined as follows :

$m_T(X) \leftarrow m1(s_1,X,[\,])$

$m1(s_1,[a|X],Y) \leftarrow m1(s_1,X,[a|Y])$ (for all a in K)

$m1(s_1,[\tilde{a}|X],Y) \leftarrow m1(s_2,[\tilde{a}|X],Y)$ (for all a in K)

$m1(s_2,[\,],[\,])$

$m1(s_2,[\tilde{a}|X],[a|Y]) \leftarrow m1(s_2,X,Y)$ (for all a in K)

Clearly $\mathbf{M_T}$ determines the mirror image language, i.e., $L(\mathbf{M_T},K\cup\tilde{K}) = \{ w\tilde{w}^R \mid w \text{ in } K^+ \}$.

(2) $\mathbf{I_T}$ is defined as follows :

$i_T([\,],[\,])$

$i_T([a|X],[a|Y]) \leftarrow i_T(X,Y)$ (for all a in T)

$i_T(X,[a|Y]) \leftarrow i_T(X,Y)$ (for all a not in T)

$\mathbf{I_T}$ is a simple projection mapping which preserves symbols from T and erases others.

(3) $\mathbf{S_P}$ is defined as follows :

$s_P(X) \leftarrow s1(q_1,X,[\,])$

$s1(q_1,[a|X],Y) \leftarrow s1(q_1,X,[a|Y])$ (for all a in $K\cup\{\not\}$)

$s1(q_1,[\tilde{\not}|X],[\not|Y]) \leftarrow s1(q_f,X,Y)$ (for all q_f in F)

$s1(q_0,[\,],[\,])$

$s1(q,[\tilde{w}^R|X],[a|Y]) \leftarrow s1(p,X,Y)$ (for all d(p,a) = (q,w))

where $A = (Q,K,D,d,q_0,F)$ is a dgsm A given in Lemma 3.1.

Then, $L(\mathbf{S_P}, K \cup \tilde{K} \cup \{¢, \tilde{¢}\}) = \{x¢\tilde{¢}\tilde{y}^R \mid f(x)=y, x \text{ in } K^*\}$

Let **P'** be a logic program defined by $p'(X) \leftarrow i_T(X,Y), m_T(Y), s_P(Y)$. It is easily seen that for x in T^*,

x is in $L(\mathbf{P},p,T)$ iff there exists y such that x = h(y) and

y is in $M_0 \cap Sp$

iff there exists y such that

$\mathbf{I_T}$ succeeds on $i_T(\mathbf{x},\mathbf{y})$,

$\mathbf{S_P}$ succeeds on $s_P(\mathbf{y})$, and

$\mathbf{M_T}$ succeeds on $m_T(\mathbf{y})$

iff **P'** succeeds on $p'(\mathbf{x})$.

Let $\mathbf{R_0}$ be defined by $r_0(X,Y) \leftarrow i_T(X,Y), m_T(Y)$.
(Since T is fixed, $\mathbf{R_0}$ is a fixed program.) Thus, p'(X) can be expressed as the desired form (3-1). □

4. What are primitives ?

We have seen in Section 3 that a specific type of logic program can play a significant role as a generator in expressing logic programs. In this section we shall examine what progrmas in the representation are primitive in more detail.

Getting back to the representation theorem, a generator program $\mathbf{R_0}$ in (3-1) of Theorem 3.1 was constructed from a weak identity program $\mathbf{I_T}$ and a logic program $\mathbf{M_T}$, i.e.,

$r_0(X) \leftarrow i_T(X,Y), m_T(Y)$

where

[0] $i_T([],[])$

$i_T([a|X],[a|Y]) \leftarrow i_T(X,Y)$ (for all a in T)

$i_T(X,[a|Y]) \leftarrow i_T(X,Y)$ (for all a not in T), and

we observe that $m_T(X)$ can be re-defined as follows :

[1] $m_T(X) \leftarrow$ append(Y,Z,X), copy(Y,Y'), reverse(Y',Z)

copy([],[])

copy([a|X],[ã|Y]) <- copy(X,Y) (for all a in K)

reverse([],[])

reverse([X|Y],Z) <- reverse(Y,T), append(T,[X],Z).

These observation leads us to the following introduction of a specific type of logic programs which can take the place of various basic programs appearing in the representation result.

Let f be a mapping from T^* to K^*. Then, consider a logic program dominated by a predicate "(f)-reverse(X,Y)", which is defined by

(f)-reverse(**x**,**y**) succeeds iff so does reverse(**f(x)**,**y**).

We call this extended reverse program. (Notice that if f is an identity, then (f)-reverse(X,Y) is an ordinary "reverse" predicate.)

Example 1.

Let f be defined by f(a)=$\tilde{a}$, f(b)=$\tilde{b}$, f(c)=$\tilde{c}$. Then, (f)-reverse (X,Y) may be, for example, defined as follows:

(f)-reverse(X,Y) <- rev(X,[ ],Y)
rev([ ],X,X)
rev([a|X],Y,Z) <- rev(X,[$\tilde{a}$|Y],Z)
rev([b|X],Y,Z) <- rev(X,[$\tilde{b}$|Y],Z)
rev([c|X],Y,Z) <- rev(X,[$\tilde{c}$|Y],Z).

Let p(X) <- append(Y,Z,X), (f)-reverse(Y,Z), then the success language of this program $\{w\tilde{w}^R | w \text{ in } \{a,b,c\}^*\}$ is context-free.

Now, let us see the next one.

Example 2.

Let f be a mapping defined by f(x) = $\tilde{x}^R$, for all x in T^*. Then, it is seen that

(f)-reverse(**x**,**y**) succeeds iff reverse(**f(x)**,**y**) succeeds
iff reverse($\tilde{\mathbf{x}}^{\mathbf{R}}$,**y**) succeeds
iff $\tilde{x}$ = y.

Let **P** be a program dominated by p(X) <- append(Y,Z,X), (f)-reverse(Y,Z). Then, the success language $L(\mathbf{P}, T \cup \tilde{T})$ is $\{w\tilde{w} | w \text{ in } T^*\}$ which is context-sensitive.

Thus, (f)-reverse can define a number of different classes of logic programs by varing a mapping f.

Now we wish to call back one's attention to the representation theorem. In the representation formula (3-1) of Theorem 3.1 a logic program can be expressed by

$p(X) \leftarrow r_0(X,Y), s_P(Y)$, where

(0) $r_0(X,Y) \leftarrow i_T(X,Y), m_T(Y)$

(1) $s_P(X) \leftarrow s1(q_1,X,[\,])$

$s1(q_1,[a|X],Y) \leftarrow s1(q_1,X,[a|Y])$ (for all a in $K \cup \{¢\}$)

$s1(q_1,[¢|X],[¢|Y]) \leftarrow s1(q_f,X,Y)$ (for all q_f in F)

$s1(q_0,[\,],[\,])$

$s1(q,[\tilde{w}^R|X],[a|Y]) \leftarrow s1(p,X,Y)$ (for all $d(p,a)=(q,w)$)

where $A=(Q,K,D,d,q_0,F)$ is a dgsm.

Let f_T be defined by $f_T(a)=\tilde{a}$(for all a in T). Then, it is easily seen that

$m_T(X) \leftarrow$ append(Y,Z,X), (f_T)-reverse(Y,Z) --- (F_1).

Further, letting f_P be a mapping defined by $f_P(x)= f(x)$, where f is a dgsm mapping induced by A, then we have

$s_P(X) \leftarrow$ append(Y,Z,X), (f_P)-reverse(Y,Z) --- (F_2).

It should be noted that for a homomorphism h, if one define a mapping f_h by $f_h(x) = h(x)^R$, then (f_h)-reverse(**x**,**y**) succeeds iff h(x)=y. Hence, a weak identity program $\mathbf{I_T}$ dominated by $i_T(X,Y)$ and involved in all the representation results is expressed by

$i_T(X,Y) \leftarrow (f_h)$-reverse(Y,X) --- (F_3)

Summarizing our argument on the use of extended reverse programs for expressing various types of basic elements in the representation result, from $(F_1),(F_2),(F_3)$ and (3-1) we obtain another representation theorem for logic programs.

Theorem 4.1(Representation Theorem 2)

Let T be a fixed alphabet. Then, there exist mappings f_h, f_T with the property that for any logic program **P** over T with a goal p(X) one can find an equivalent logic program **P'** with a goal p'(X) such that it is expressed by

$$p'(X) \leftarrow (f_h)\text{-reverse}(Y,X), \text{append}(Z_1,Z_2,Y), (f_T)\text{-reverse}(Z_1,Z_2),$$
$$\text{append}(W_1,W_2,Y), (f_P)\text{-reverse}(W_1,W_2),$$

for some mapping f_P.

5. Concluding Remarks

Through the formal language theoretic formulation, we have shown two representation theorems for logic programs. First, we introduced the concept of the success language of a logic program, and associating a logic program with its success language we gave a formal language theoretic semantics of logic programs.

Further, using the language theoretic semantics a representation theorem for logic programs was provided in which some type of fixed logic programs play the role of a generator in the representation.

Then, it has been considered the problem of what operation is primitive for the representation of logic programs. By introducing the concept of an extended reverse predicate, it has been proved that one need only "append" and "extended reverse" functions in representing logic programs.

For the future research in this direction, using a model-theoretic semantics in terms of the success language one may discuss many issues on the properties of a logic program such as program transformation, program classification, program synthesis, and so forth, some of those which we are going to work on.

For example, in the context of program synthesis, we now have an effective (automatic) procedure for obtaining a logic program from a program specification given in formal grammatical formulation which is one of the direct applicatons of the representation result.

ACKNOWLEDGEMENTS

The author is indebted to Dr.Tosio Kitagawa, the president of IIAS-SIS, Dr.Hajime Enomoto, the director of IIAS-SIS, for their useful suggestion and warm encouragement.

He is also grateful to his colleagues, Toshiro Minami,Taishin Nishida who worked through an earlier draft of the paper and suggested the present improved formulation.

REFERENCES

[1] Colmerauer, A., Les systemes-Q ou un formalisme pour analyser et synthetiser des phrases sur ordinateur, Internal publication no.43, Dept. d'Informatique, Universite de Montreal, Canada, September, 1970.

[2] Chomsky,N.and Schutzenberger,M.P.,The algebraic Theory of context-free languages,in"Computer Programming and Formal Systems (Braffort and Hirschberg, eds.), North-Holland, Amsterdam, 118-161, 1962.

[3] van Emden,M.H. and Kowalski,R.A., The Semantics of Predicate Logicas a Programming Language, JACM 23 : 733-742 (1976).

[4] Engelfriet,J. and Rozenberg,G., Fixed Point Languages, Equality Languages, and Representation of Recursively Enumerable Languages, JACM 27: 499-518 (1980).

[5] Harrison,M.A., Introduction to Formal Language Theory, Addison-Wesley, 1978.

[6] Hopcroft,J.E. and Ullman,J.D., Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

[7] Kowalski,R., Predicate logic as a programming language,in "Proc. of IFIP-74", 569-574.

[8] Salomaa, A., Formal Languages, Academic Press, 1973.

[9] Shapiro,E.Y., Alternation and the Computational Complexity of Logic Programs, J. of Logic Programming 1: 19-33 (1984).

[10] Tarnlund, S.A., Horn Clause Computability, BIT 17: 215-226 (1977).

[11] Yokomori, T., A Logic Program Schema and Its Applications, in "Proc. of 9th IJCAI", UCLA, CA, Aug., 723-725, 1985.